Dangers of creative writing

Creative writing happens when some routine that accepts a variable name in order to modify its value gets creative about how to resolve the name, and ends up using the global namespace instead of the current one.

See Also

clock scan invalidates variable
: clock scan falls victim to creative writing.

Description

If multiple namespaces are considered when resolving an identifier, the identifier might not be resolved as intended. Creative writing is not unique to Tcl. In ECMAscript, for example, creative writing happens when the keyword var is ommitted, causing a variable to be assigned on the global object rather than in the immediate context. In Tcl, creative writing of a variable does not occur in the body of a procedure because each variable is resolved only in the local procedure body. For a script evaluated in a namespace, creative writing of a variable may occur because both the current namespace and the global namespace are searched. For routines, the path may include any number of namespaces, making the conditions for creative writing dynamic.

The following Tcl **commands** are ***creative writers***:

- append
- array set
- binary scan
- catch
- dict append|create|update|...
- file stat
- file lstat
- foreach
- gets
- incr
- info default
- lappend
- lassign
- regexp
- rename
- regsub
- scan
- set
- try
- unset

When code is evaluated in a **namespace** (not in a procedure, but directly in a **namespace**) the **namespace** rule for name resolution applies:

> *If you provide a fully-qualified name that starts with **::**, there is no question about what command, variable, or namespace you mean. However, if the name does not start with **::** (i.e., is relative), Tcl follows a fixed rule for looking it up: Command and variable names are always resolved by looking **first** in the **current namespace**, and **then** in the **global namespace**.* (namespace documentation)

This leads to dangerous context-dependency: depending on what global variables **exist**, new variables are created either **here** or **there**. Ronnie Brunner wrote in comp.lang.tcl:

```
% set a global
global
% namespace eval foo {set a "namespace"; set b "namespace"}
namespace
% set a
namespace
% set foo::a
can't read "foo::a": no such variable
% set b
can't read "b": no such variable
% set foo::b
namespace
```

Hemang Lavana commented: To be safe, you should always use variable to first create a variable in the current namespace.

RS: But still, this requirement for a declaration is not in the spirit of Tcl.

In contrast to set, proc resolves a procedure relative to the current namespace, even if a procedure in the global namespace by that same name already exists, so proc is **unaffected** by the *creative writing* problem:

```
% proc a {} {puts global}
% namespace eval foo {proc a {} {puts namespace};proc b {} {puts namespace}}
% a
global
% foo::a
namespace
% b
ambiguous command name "b": binary break
% foo::b
namespace
```

PYK 2015-12-28: proc may not be a creative writer, but rename is:

```
namespace eval one {
    rename puts zork
}
puts hello ;#-> invalid command name "puts"
```

RS: I think namespace resolution should distinguish between locating existing things (procs or variables) and determining an absolute name for non-existing things.

- For locating existing things, the current resolution rule can be left unchanged.
- For "creative writing", creating new things, it should confine itself to the current namespace (as it currently does with procs). He who wants something done in the global namespace, can explicitly prepend :: to the name ;-)

NEM 2005-09-09 PYK 2020-10-09: Just looking at this old discussion again, a pertinent question would be what should be the appropriate behaviour in this situation:

```
set a 12
namespace eval foo {set a [expr {$a+1}]}
```

Here we apparently have the same variable referenced twice, but what should be the effect of the second statement? There are a number of possibilities:

1. ::a (global) gets set to 13. This is the current behaviour of set.
2. ::a is left as 12, ::foo::a gets set to 13. This is the behaviour of variable.
3. error "no such variable 'a' while executing 'expr {$a + 1}'". A variable is resolved only in the current namespace.

There are probably a few other options, too. Of these, both 2 and 3 seem much better than option 1. I used to think option 2 would be the best (which looks like RS's suggestion, too), but option 3 might be the clearest one of all, and is the behaviour that we get in proc bodies now (all vars are local for read or write unless explicitly qualified or imported).

EKB: I'd vote for #3! In the case where the definition of a outside my namespace is far removed from where I'm using it, behaviors #1 & 2 could be very confusing.

Update on 2007-01-16: Ralf Fassel pointed out another *creative writer* in comp.lang.tcl:

```
checkbutton .foo.bar.baz
```

will create/use a global variable 'baz' which might have surprising effects if 'baz' is already in use as global variable with a different meaning elsewhere in the program.

LV: I wonder if you investigate Tk, whether you will find more of these.

RFox: I think radiobutton does similar stuff.

DKF: The rule is "*always* set the -variable for a checkbutton or radiobutton".

AK: So why not make this option required ? (Is that actually possible in Tk ?)

DKF: You've just answered why. There is no mechanism in the current option management code to make any option required during object creation (which is the only time you'd ever want it to be mandatory anyway).

TIP 278 would take away the namespace problems with respect to creative writing, but apparently not the checkbutton problem.

AMG: Curious why [unset] qualifies as a creative writer. I can see that its behavior is context-dependent, same as the others, but it will not create a variable.

PYK 2015-03-22: Other creative writers don't necessarily create a variable either. It's when they don't that the creative writing effect is pronounced. The main problem with creative writing is that the command operates on a variable other than the one the author expected . A script like

```
% unset -nocomplain somevar
```

might leave a variable in the [global] **namespace** in an unexpected state, irrespective of the intentions of the author.

AMG: I see what you mean:

```
% set x hello
% namespace eval foo {unset -nocomplain x}
% set x
can't read "x": no such variable
```

```
% set x hello
% namespace eval foo {variable x goodbye; unset -nocomplain x}
% set x
hello
```